

## 08 OOP - その2

作成者 : GT F

最終更新日 : 2019-08-13

- [継承](#)
- [UML](#)
- [IS-A 関係](#)
- [override](#)
- [super & this](#)
- [型判断](#)
- [Object](#)
  - [toString](#)
  - [equals](#)
- [継承&型変換 \(キャスト\)](#)
- [集約関係](#)
- [継承不可](#)
- [質問](#)
- [覚えておこう](#)

### 継承

クラスの記述するときは基底クラス (base class) を指定できます。派生クラスは基底クラスから導出されます。クラスは**単一継承**のみに対応します。

 派生クラスは public / protected 属性のみアクセス可能。

派生した (引き続く方) クラスは「サブクラス」と呼びます。引き続がれた方は「スーパークラス」あるいは「親クラス」と呼びます。キーワード : `extends` を使用して、クラスを継承します。

```
1 // Parent.java 親クラス
2 public class Parent {
3     /** 姓 */
4     public String familyName;
5     /** 名 */
6     private String givenName;
7     /** 財産 */
8     protected int money;
9     /** メソッド */
10    public String hello() {
```

```

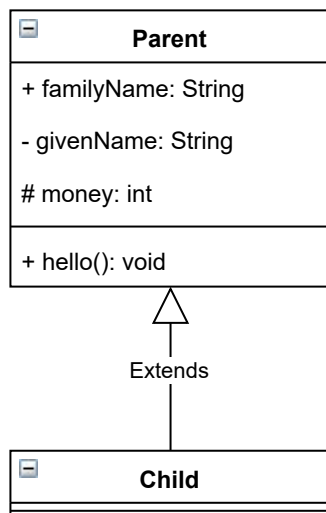
11     System.out.println("say hello");
12     return "123";
13 }
14 }
15 // Child.java サブクラス
16 public class Child extends Parent {
17
18 }
19 // Kicker.java
20 public class Kicker {
21     public static void main(String... args) {
22         Child child = new Child();
23         System.out.println(child.familyName);
24         // String value = child.hello();
25         // System.out.println(value);
26         System.out.println(child.hello());
27     }
28 }

```

質問：Parentクラスの3つフィールドを保有しています。Childクラス2つフィールドを継承します。親クラスのgivenNameはprivateの為、継承しません。

## UML

UML Class図に、継承の記載方法



## IS-A 関係

クラスBがクラスAを継承する場合、日本語で「BはAの一種である」を成り立つ。クラスCがクラスBを継承するのであれば、「C is B」だけではなく、「C is A」も成り立ちます。このような文で表せる関係は「IS-A」関係と呼び出す。

```
1 // Shape.java 図形
2 public class Shape {}
3 // Square.java 正方形
4 public class Square extends Shape {}
5 // Kicker.java
6 public class Kicker {
7     public static void main(String...args) {
8         Shape p = new Shape();
9         Shape c = new Square(); // 正方形は図形の一種である
10    }
11 }
```

## override

オーバーライドとは、サブクラス内で、スーパークラスで定義しているメソッドを同じ名称に再定義することです。サブクラスがインスタンス化され、オーバーライドされたメソッドが呼ばれた際には、サブクラスに再定義されたメソッドを優先的に呼び出される。

```
1 // Parent.java
2 public class Parent {
3     public void hello() {
4         System.out.println("parent method");
5     }
6 }
7 // Child.java
8 public class Child extends Parent {
9     @Override // 親クラスの強制的にOverrideする意味 (アノテーション)
10    public void hello() {
11        System.out.println("child method");
12    }
13 }
14 // Kicker.java
15 public class Kicker {
16     public static void print(Parent v) {
17         v.hello(); // Override 実際発生する箇所
18     }
19     public static void main(String...args) {
20         print(new Parent()); // parent method
21         print(new Child()); // child method ⇒ 派生クラスは直接に渡す可能
22     }
23 }
```

## super & this

`this` は、自分の**インスタンス**を指すキーワードです。`super` は親クラスのクラスメンバを指しています。

```
1 // Parent.java
```

```
2 public class Parent {
3     protected int b; // 注意
4     private int a;
5     public void hello() {
6         System.out.println("parent method");
7     }
8 }
9 // Child.java
10 public class Child extends Parent {
11     protected int b; // 注意
12     public Child() {
13         super(); // 親クラスのコンストラクタを呼び出す
14     }
15     @Override
16     public void hello() {
17         super.b = 100;
18         this.b = 200;
19         System.out.println("invoked from child");
20     }
21 }
```

## 親クラスのメソッドを呼び出す

```
1 // Parent.java
2 public class Parent {
3     public void hello() {
4         System.out.println("parent method");
5     }
6 }
7 // Child.java
8 public class Child extends Parent {
9     @Override
10    public void hello() {
11        super.hello(); // 親クラスのhello()メソッドを最呼び出す
12        System.out.println("child method");
13    }
14 }
15 // Kicker.java
16 public class Kicker {
17     public static void main(String...args) {
18         Child c = new Child(); // cのタイプはChild又はParentであること
19         c.hello(); // child method ⇒ 派生クラスは直接に渡す可能
20     }
21 }
```

## 型判断

型の判別は演算子 `instanceof` を使用して判別します。

演算子	説明
instance of	あるオブジェクトは指定するクラス型であるかを判断する

実際 `instanceof` を使用例です。

```

1 public static void main(String[] args) {
2     Parent p = new Parent();
3     Child c = new Child();
4     System.out.print(p instanceof Child); // false
5     System.out.print(c instanceof Parent); // true
6 }
```

## Object

全てのクラスはObjectクラスを継承しています。このクラスはJava参照型（クラス型）を基本機能を定義しています。よく以下2つメソッドをオーバーライドします。

1. `toString()` : オブジェクトの文字列の表現を返す。
2. `equals(Object another)` : 引数のオブジェクトは等し（ひとしい）かを返す。

## toString

インスタンスを文字列に変換して返します。`toString()`をオーバーライドしない場合、インスタンスのHashCode（メモリアドレス）を返却します。この場合人間は読めません。以下例を実施してください。

```

1 public class Kicker {
2     public static class InnerClassA {
3         public String name;
4         public InnerClassA(String name) {
5             this.name = name;
6         }
7     }
8     public static class InnerClassB {
9         public String name;
10        public InnerClassB(String name) {
11            this.name = name;
12        }
13        @Override // ObjectクラスのtoString()をオーバーライドする
14        public String toString() {
15            return "クラス名称=" + name;
16        }
17    }
18
19    public static void main(String... args) {
20        // ?? Object.toString()を呼び出す
```

```

21      System.out.println(new InnerClassA("innerClassA"));
22      // ?? InnerClassB.toString() を呼び出す
23      System.out.println(new InnerClassB("innerClassB"));
24  }
25  }

```

## equals

演算子 `==` はオブジェクトを比較し、同じメモリアドレス情報が一致するかを判断します。

```

1  String str1 = new String("string 1");
2  String str2 = new String("string 1");
3  System.out.println(str1 == str2); // false: メモリアドレスは一致しません。

```

`equals` メソッドは各クラスでオーバーライドすることができます。Stringクラスは `equals` をオーバーライドして、メモリアドレス比較ずに、文字列の内容を比較します。

```

1  String str1 = new String("string 1");
2  String str2 = new String("string 1");
3  System.out.println(str1.equals(str2)); // true: 文字列内容は一致します。

```

## 継承 & 型変換（キャスト）

親クラス⇒子クラスキャストできる前提条件は該当オブジェクトの実型はキャスト対象クラス

```

1  public static void main(String... args) {
2      Object str = new String("DCNET");
3      String value1 = (String)str;
4      String value2 = String.class.cast(value1);
5      String excep = (int)str; // エラー: 変数strの実型はStringである。
6  }

```

## 集約関係

あるオブジェクトが他のオブジェクトの一部であったり、他のオブジェクトをもっていたりする関係を**集約**といいます。アマゾンの各商品にて写真が保有しています。各写真は大小3サイズの画像を保存しています。以下通りクラスを定義する。

```

1  public class Image {
2      private String minImage;
3      private String middleImage;
4      private String largeImage;
5      // getter setter
6      // ....

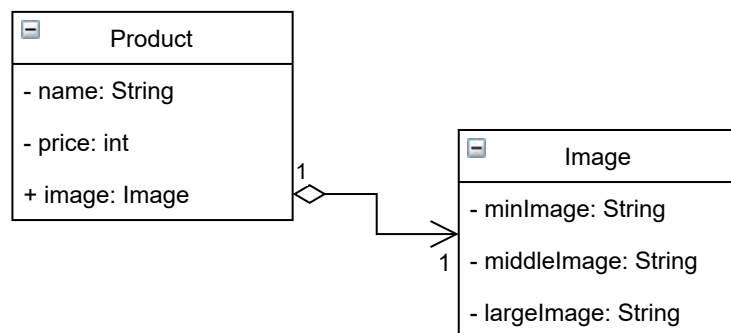
```

```

7  }
8  public class Product {
9      /** 名称 */
10     private String name;
11     /** 価格 */
12     private int price;
13     /** 商品画像 */
14     private Image image;
15     // getter setter
16     // ...
17 }

```

UMLの表現：ProductがImageを「**所有する**」。



## 継承不可

`final` キーワードが指定された場合、該当クラスは継承不可になります。

```

1  public final class NoChild {} // 該当クラスを継承不可

```

質問：以下コードのコンパイルエラー理由を教えてください。

```

1  // Parent.java
2  public final class Parent {}
3  // Child.java
4  public class Child extends Parent {}

```

## 質問

質問 1：以下コードの出力するを回答してください。

```

1  String x = "a";
2  String y = "a";
3  System.out.println(x == y); // true ? false? 原因は ?
4  System.out.println(x.equals(y)); // true ? false? 原因は ?
5

```

```
6 String empty = "";
7 String nullString = null;
8 System.out.println(nullString.equals(empty)); // true ? false? 原因は ?
```

質問2 : 以下コードの出力を回答してください。

```
1 // Employee.java
2 public class Employee {
3     private String id;
4     public Employee(String id) {
5         this.id = id;
6     }
7     @Override
8     public boolean equals(Object another) {
9         if (another instanceof Employee) {
10             Employee an = (Employee) another;
11             return an.id.equals(id);
12         }
13         return false;
14     }
15 }
16 // Kicker.java
17 public class Kicker {
18     public static void main(String...args) {
19         Employee x = new Employee("1234");
20         Employee y = new Employee("1234");
21         System.out.println(x == y); // true ? false? 原因は ?
22         System.out.println(x.equals(y)); // true ? false? 原因は ?
23     }
24 }
```

質問3 : 以下コンパイルを修正してください。

```
1 // Parent.java
2 public class Parent {
3     public void hello() {
4         System.out.println("parent method");
5     }
6 }
7 // Child.java
8 public class Child extends Parent {
9     @Override // 親クラスの強制的にOverrideする意味 (アノテーション)
10    public void hello() {
11        System.out.println("child method");
12    }
13 }
14 // Kicker.java
15 public class Kicker {
16     public static void print(Child v) {
17         v.hello();
18     }
19     public static void main(String...args) {
20         print(new Parent());
21     }
22 }
```



```
21         print(new Child());  
22     }  
23 }
```

## 覚えておこう

 いいね 1 番に「いいね」しましょう

ラベルがありません 